This document contains evidence for the development of the first prototype. This will include the iterative development of the code as well as notations of any errors/corrections and testing done during the development of the prototype.

**Setting Up Unity**The Unity engine environment allows you to organise all assets such as code, materials and game objects in folders and set up what is associated with each other. The environment also allows the user to organise their ‘scenes’ – these are basically the blank canvas worlds to build games upon. The first part of the development for the prototype is to set up the folders and scenes for it.

As seen in, I have organised a parent assets folder. This parent folder contains sub-folders which will contain such things as scripts, materials, scenes and miscellaneous components. This organisation will not affect the code of the prototype, instead this is there for the sake of organisation and easy access for me. It will make the development process a lot more efficient.

**Code Development – Message Bus**The first set of code that I implemented into my prototype is a *‘Message Bus’* script. This script was added in as a way of being able to broadcast information across the entire prototype without leaving methods, variables and even classes in public and having being able to be accessed by anything else. This will allow me to control passing of information a lot easier, rather than risking information being modified by other parts of code.

Error Review – Unity Implementation  
When I initially added the message bus code into the prototype, the message bus instantly gained errors. The code wasn’t initially built for it to work in Unity and as such I was expecting errors but none of this scale. The main issue that occurred with the implementation was from terms within the code. Unity uses the term *‘Object’* for physical game objects and as such, the engine was confused as *‘objects’* in the message bus weren’t game objects in the unity environment.  
The solution to the error was relatively simple. I required to indicate into the code that the term *‘Object’* in the *‘Message Bus’* code wasn’t referring to a game object in the Unity engine – as it is by default; using *‘System.object’* refers it as part of the class written rather being a game object within the Unity engine.

**Prototype Development – Procedurally Generated Map**The next part of the code I will start to develop will be the procedurally generated map. This is arguably the most important part of this first prototype and also the longest. As this generation will be made up of multiple game objects in the Unity engine, this set code will take up two scripts.

**Stage One – Building Array**  
The procedural generation works from an array and so the first part would be to produce the array and fill it with information for the procedurally generated map. This would be the first representation of a generated map and with this successful, would allow me to move on to further development.

First lines of code define the important variables that are needed at the start, this is such things as the dimensions of the generated map, the empty variable that would be used for the seed and the Boolean option to use a randomly generated seed (currently the seed for this is set to the date & time set on the computer system it is being ran on). This information on what these lines of code are for are also indicated through the use of comments, the comments currently lay out what a chunk of code is for – example would be for commenting what the purpose of a particular function is.

Error Review – Instance of an Object  
When implemented in Unity, when told to play, nothing happened – nothing spawned nor did any compiling errors happened. When looking at the Unity engine console the error message was present:  
*‘NullReferenceException: Object reference not set to an instance of an object  
MapGeneratorScript.FillMap () (at Assets/Scripts/MapGeneratorScript.cs:56)  
MapGeneratorScript.BuildMap () (at Assets/Scripts/MapGeneratorScript.cs:39)  
MapGeneratorScript.Start () (at Assets/Scripts/MapGeneratorScript.cs:32)’*

The issue is unlike most other errors, this is a logical error – it doesn’t effect the system in it’s ability to compile it, but the code I have written does not work as expected.  
I made an error when programming, using private variables but never implementing *‘Setters & Getters’* meaning that none of the variables were actually being accessed by any of the methods. I intitally thought this was only a problem with the variables *‘mapSeed’* & *‘randomGeneratedSeed’* and so I adujsted accordingly.

This unfortunertly didn’t sort out the problem; I did however realise that I haven’t been using *‘Setters and Getters’* across the entire code and I also reconciled that issue. I decided however to make the variable *‘wallDensity’* fully public – this is only temporary and is to allow me to make a choice on the space within the generated map; once I come to a happy decision I will set it as my desired number and lock it.

With those put in place this problem was partly sorted. The code when run in Unity generated something in the enviroment, however not as I wanted.

Error Review – No Seed  
As you can see in the image on the left, the gizmo function is drawing something – but it is all purple. This means, according to the gizmos function, all of the fields in the array are 0 i.e floors. This is a problem with the seed generation, the fact that a seed has not actually been generated.

The solution however came from a different place than I thought; a line of code that is supposed to implement the random seed if selected was written completely wrong – instead of filling the map in accordance to the wall density, it was filling up the map on it’s own. This cause the generated map to be fully floor or fully wall for reasons that I am still not sure about. The code has now been rectified and compiles with the wall density.

Testing – Gizmo Drawing  
In order to see whether the generated map array was being correctly built, I created a method that would be able to draw a visual representation of the array. This method of Gizmo drawing will present all that would be a wall in the colour black and all that would be a floor the colour magenta.  
This test will consist of generating multiple map, using different seeds and repeats of seeds; this will be able to assess any anomolies with any of the map generations and also see any issues with prolonged map generation. After multiple tests, I would say that the map generation is a success. Seeds written generate different maps and the percentage of the walls match the ratio of wall to floor. A lot of map generations however consisit of isolated rooms and isolated wall sections within reigons; being unable to access the isolated reigons is a waste of processing – I would rather create passageways or remove them all together.

**Stage Two – Starting Mesh Builder (Marching Squares)**With the array operating, I have decided to continue on and start creating the basis of the mesh builder – with the alterations to the map generation in the array coming later. If I have both up and running in some sort of functioning state, I will be able to work on them simultaniously.  
The main chunk of the mesh builder is via the inclusion of the marching squares logic for creating smoother ‘cave-lite’ walls. This consists of creating object *‘nodes’* to act as points of squares (the squares being each field in the array and therefore each tile in the map) to create triangles from – this will eventually build up the walls and corners in the physical enviroment.

Error Review – Protection Levels  
When writing the code for this prototype and project in general, I have been trying to write classes, methods and variables all in private – with having *‘setters & getters’* being used and using the message bus to pass information across when needed. However, when writing the *‘MeshBuilder’* C# script, the system will not work at all. The script will not even be compiled and as such will not run.

Even with additions of *‘Setters & Getters’* included, the main systems of the *‘MeshBuilder’* including the Control Nodes and the square drawing gizmos would not compile and was riddled with errors. At this point in time, I have found that the *‘Mesh Builder’* script to be written entierly in public found it to work correctly as the script can access any part of itself thereby having access to all of the data. While this isn’t the best decision as it leaves to issues such as overwriting if the code is incorrectly written, it allows the *‘Mesh Builder’* script to compile and run properly. I will have to come back on this issue and see if I can move the methods & variables of the script to be private and access them in a alternate way.

Error Review – Double Gizmos  
To see whether the actual *‘Mesh Builder’* script is functioning properly, I included another Gizmos method that draws out the squares and nodes defined in the *‘squareGrid’* and *‘Sqaures’* classes and methods. If done correctly, the Gizmo should create the map grenerated from the array built in the *‘MapGenerator’* script – floor would be white squares while walls would be black squares, with nodes being grey.

While this did happen, therby the Gizmos method being a success, I forgot to factor in that there are two Gizmo methods running at the same time – the Gizmos method written in the *‘MapGenerator’* script was still active at the time. This caused a large workload for the computer system that I was using as it was essentially live-rendering many squares – and doing this twice. This was a simple fix of commenting out the first Gizmo method I wrote, there by not being compiled when told to run.

It was good in a sense to have both of these Gizmos running at the same time – it showed that the Gizmo in the *‘MeshBuilder’* script was written correctly and took the information from the *‘MapGenerator’* script.

Testing – Mesh Gizmos  
With the Gizmo method included into the *‘MeshBuilder’* script, I tested the Gizmo drawing the same way I did with the Gizmos method written in the *‘MapGenerator’* script. Like before, I ran the scene multiple times with multiple seeds to see if any anomolies occurred during map generation. As seen in the image, the map is being built in the same manner as when shown via the original Gizmo method generation, meaning that no error in information transfer occurred.

**Stage Three – Building the Mesh**With the basis of the mesh builder now in place, I can start to actually build the physical mesh and create the 3D walls for the map generation. At this point, the mesh will not have any collision dection with other game objects in the world nor some sort of floor or alterations but having this implemented now would mean I will not have to worry about issues with the walls or physical rendering if I include any more rules into the map generation itself.

This requires more additions to the *‘Mesh Builder’* script by adding a method which generates the triangles from the nodes created from the last stage of development. What triangle will be generated for that particular tile of wall will depend on a number of factors and as such the code will include different **cases** from which to decide what triangle(s) will be rendered.

Error Review – No Components  
After the marching squares code was implemented, I went to Unity to test whether the mesh was being created. I expected a 2D mesh to be spawned in the centre of the scene – but instead nothing spawned. The code was able to properly compile and as such it was rather difficult to point out what was the particular problem with the code. The error given to me by Unity was that the *‘Mesh Filter’* component was not added to the Map Generator game object even though this was not the case.

Turns out the problem was that I missed out on some code. While I implemented the rules of the marching squares algorthim, I did not actually put in code to tell the Unity game object to such things. This lack of code rendered the *‘MeshBuilder’* script useless.

Testing – Mesh Mid-Way Test  
The *‘MeshBuilder’* script now creates a mesh plane and in replacement of the Gizmos method, I can use this as a test for map generation to find any anomolies. At this time of development, I have not added any new rules for the map generation and it is still in it’s basic state so looking for anomolies in the map generation is rather redundant. I just want to make sure that in this mid-test that the mesh renders properly.

When performing these tests, I noticed that there was a lot of floor space compared to wall space. I intitally set the ***wallDensity*** variable to be 47 (47% of the tile map will be wall) but it seems that this value is too low. I have now set the value of this variable to be 50. This should provide a more closed off cave generation which is more prefered for the final product that I am creating.

With the 2D mesh put in place, I can now incorporate the the physical walls into the mesh and create a 3D mesh. With this done, that will mean that apart from the collision dectection, the mesh will be completed and major development on the *‘MeshBuilder’* script can end.

The addition to make the mesh 3D is just adding a thrid-dimension to the mesh. New methods are needed that work in a similar way to the ones used to make the 2D mesh intially, but these instead use an addition of an integer to incorporate the walls. In Unity, the addition of a new mesh will require a new game object to hold a *Mesh Filter* and *Mesh Renderer* for it – this will be incorportated as a child of the *‘Map Generator’* game object so they will be associated with each other within Unity.

With that code implemented, the game object in Unity now generates a three-dimensonal mesh. It doesn’t however, generate a floor. In order to build a floor I have to include another game object into Unity known as a plane. I intially wanted to include the plane as a *‘preFab’* but considering that the generated map is of a specific size that won’t change during gameplay, I decided to just leave the object static and add it to the bottom of where the map will be generated.

**Stage Four – Map Modifications**With the physical elements (minus the collision dectection) implemented into the map generation. The generated map’s mesh is finished – this will only be returned to for any edits or slight modifications I would like to make during development of other things or from feedback. The stage of development I’ll be focusing on will be the addition of map generations rules for map modifaction.

The basic map generation contains closed off cave areas, small anomolies within the cave and larger areas closed off from each other. I will need to remove small anomolies and tiny passage ways but I do however want to include passageways to connect each of the larger maps.

Code Development – Room Removal  
Within the current generated map, there are multiple generations which create small wall spaces or room spaces. These spaces are unfortunertly too small to create any rooms out off or have a wall mass in the middle of the room so it will be best to remove them all together. The algorthim I have written identifies regions created in the generated map before removing all regions that are less than a determined minimum size. Currently I have set the minimum size to be 50 – this size keeps rather small spaces but removes anything fairly tiny, allowing me to create small rooms and entranceways.

Struct Addition – Co-Oridinates  
To identify each of the regions made in each generated map, I require a way to trace the location and sizing of each region. To do this, I included a new struct in my code to create co-ordinates; called *‘Coord’* within my code, it uses new variables for x-axis and y-axis. This can be implemented into other methods to create co-ordinates from different values. Because multiple methods require access to this to create co-ordinates of their own, the variables within this code is currently sent to public but the struct itself however has been set to private.

Mid Testing – Space Removal  
With this addition of code implemented, I performed some testing to see how this inclusion of anomoly removal. Using multiple seed generation I took a look on how the map generations turned out post-implementation. I got multiple people from the focus group to input statements of their choice to be used as seeds as well as use the random generator seed – this still uses the time set on the computer.  
Overall, I feel that the new generation is a solid improvement. It easily removes tiny rooms or walls within room space but still leaves out some small rooms for me to include passage ways for. It can still get a bit clustered and fragmented such as the generation shown in figure 34 on the top left. These can still be connected with passageways but I may possibly minimum area size for a room/floor area.

Code Development – Passageways  
With the area removal code complete, I can now implement the generation of passageways. This includes the addition of methods to match up each of the rooms together as well as the addition of a method to essentially carve-out tunnels between each rooms. I also used this opportunity to include summarys into my code and replace all of the manually imputted comments.

Method Addition – Gradients  
As my information for map generation works essentially on a grid, to create the connection lines, I implemented a method that calculates gradient lines between rooms. Using the co-ordinates struct, it uses the equation of a straight line to create the desiered pathway in the map generation.

Testing – Mesh Generation  
With the code for the passageway implemented, that would make the map generation part of my prototype complete. Using some seed names and the randomly generated seed, I gave the mesh generation another small development test. The implementation of the passageway code has made the caves being generated a lot more complex, with tunnels, open areas and tight rooms incorportated into the majority of generations which is what I was looking for in my map generation. I will consider the whole map generation to be a success.

**Prototype Development – Menu Implementation**With the map generation now complete, I can start working on the next main part of my first prototype which is the starting menu. The plan for the menu is to have it allowing the user to input a seed from the menu or toggle to start the generation with a random seed before having a button to generate the map with. This will also provide the option to quit from the prototype which will be needed as I will be doing the main testing with my peer group via a build application.

Setting Up Unity – Menu Objects  
Creating a menu for my first prototype requires a couple of additions to it within the Unity editor. The first was the addition of a canvas; a canvas is where the physical elements of the user interface are placed, working as a parent object to object children. The second addition to this was a new folder within the Unity files *‘fonts’*. Unity doesn’t allow the user to access the fonts installed onto the computer, as when a game is built, not all computers may have access to those fonts – I will need to aquirre some fonts to use for text elements and the files shall live in this folder. Before I start writing code, I added the majority of my physical elements to the menu first – this includes the tile and the play & exit buttons.

Method Addition – Menu Function  
The menu requires a new script for the functionality of the menu. This script contains all the methods for the main menu, including quitting the application and starting the map generation scene. The methods themselves are reletively small, containing a maximum of about 4 actions to execute. This is not suprising given that this is only the first prototype, this will grow to be more complex via the other prototypes.

Error Review – Script Communication  
A major part of my procedurally generated map and a major part of my main menu is the feature to incorporate the ability to allow users to input their own seeds or have the option to toggle using a randomly generated seed. My intial plan was to have the values *‘mapSeed’* and *‘randomGeneratedSeed’* changed via UI elements but it was wasn’t able to directly change the values due to the fact that the methods for the UI elements and the values that want to be changed are in separate scripts. I tried to change the the values being changed to statics – this allows for the values to be changed for any script. However, it is a reletively bad idea to use statics. Instead, I created a manager method that allows me to change the values without being them set to static. This however still didn’t work. I assume that the reason for these failed fixes is due to the fact that the map generator script isn’t running. The menu script is used within a different scene compared to the map generation script – this makes me assume that the values aren’t being changed simple cause the values *don’t exist.*

Solution – File Management Class  
It was concluded that simply changing the values of the string and bool will not work, this action wouldn’t simply pass through as the *‘MapGeneratorScript’* isn’t actually active at the time. I thought of another method to change the values of the string and bool without using a direct method – via a file. If I create a file, I would be able to load whatever is in the file as a string in the *‘MapGeneratorScript’* and use that as a seed as the file will be saved when made in the *‘MenuScript’*. However, simply adding a method to create a file wasn’t working either; while it was being compiled, a file wasn’t being created. After some time I came to a conclusion to fix this problem – making another class script. This class, now known as *‘FileWorker’*, manages the files made for the map generation, containing the methods for it – these mehtods are then used in the other scripts for it to function. This is a lot different than what I planned out for my design as I intially thought that this could all be done via the two intial scripts.

**Code Development – Viewing Camera**At this point, the main development of the prototype is complete but there is one final addition that I need to incorporate which is a camera. The camera is simply there to provide a better viewing of the generated map, which will allow better assessment of any anomolies within it. This is just a small amount of code in the form of a new script, known as *‘ObritalCamera’* which includes a method that makes the main camera within the *‘Generation Scene’* rotate around the y-axis of the generated map.

**Development Completed – Prototype One**With this final camera put in place, I feel that the prototype is now completed. With that comes for final tweaks being made and a final in-development test. The tweaks that I made included changes to the orbital camera angles and edits to the positions of the UI elements on the canvas. I also did some cleaning up of my Unity editor and folders to make organisation easier.